Tradeoffs between synchronization, communication, and work in parallel linear algebra computations

Edgar Solomonik
Erin Carson
Nicholas Knight
James Demmel

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-8
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-8.html

January 25, 2014

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
25 JAN 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Tradeoffs between synchronization, communication, and work in parallel linear algebra computations	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Electrical Engineering and Computer Sciences,Berkeley,CA,94720**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper derives tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost. Our theoretical model counts the amount of work and data movement as a maximum of any execution path during the parallel computation. By considering this metric, rather than the total communication volume over the whole machine, we obtain new insight into the characteristics of parallel schedules for algorithms with non-trivial dependency structures. The tradeoffs we derive are lower bounds on the execution time of the algorithm which are independent of the number of processors, but dependent on the problem size. Therefore, these tradeoffs provide lower bounds on the parallel execution time of any algorithm computed by a system composed of any number of homogeneous components each with associated computational, communication, and synchronization payloads. We first state our results for general graphs, based on expansion parameters, then we apply the theorem to a number of specific algorithms in numerical linear algebra, namely triangular substitution, Gaussian elimination, and Krylov subspace methods. Our lower bound for LU factorization demonstrates the optimality of Tiskin?s LU algorithm [24] answering an open question posed in his paper, as well as of the 2.5D LU [20] algorithm which has analogous costs. We treat the computations in a general manner by noting that the computations share a similar dependency hypergraph structure and analyzing the communication requirements of lattice hypergraph structures.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**19**	

Tradeoffs between synchronization, communication, and work in parallel linear algebra computations

Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel

Department of EECS, University of California, Berkeley

January 25, 2014

Abstract

This paper derives tradeoffs between three basic costs of a parallel algorithm: synchronization, data movement, and computational cost. Our theoretical model counts the amount of work and data movement as a maximum of any execution path during the parallel computation. By considering this metric, rather than the total communication volume over the whole machine, we obtain new insight into the characteristics of parallel schedules for algorithms with non-trivial dependency structures. The tradeoffs we derive are lower bounds on the execution time of the algorithm which are independent of the number of processors, but dependent on the problem size. Therefore, these tradeoffs provide lower bounds on the parallel execution time of any algorithm computed by a system composed of any number of homogeneous components each with associated computational, communication, and synchronization payloads. We first state our results for general graphs, based on expansion parameters, then we apply the theorem to a number of specific algorithms in numerical linear algebra, namely triangular substitution, Gaussian elimination, and Krylov subspace methods. Our lower bound for LU factorization demonstrates the optimality of Tiskin's LU algorithm [24] answering an open question posed in his paper, as well as of the 2.5D LU [20] algorithm which has analogous costs. We treat the computations in a general manner by noting that the computations share a similar dependency hypergraph structure and analyzing the communication requirements of lattice hypergraph structures.

1 Introduction

We model a parallel machine as a network of processors which communicate by point-to-point messages. This model has three basic architectural parameters,

- α – network latency (time) for a message between a pair of processors,
- β – time to inject a word of data into (or extract it from) the network,
- γ – time to perform a floating point operation on local data,

which are associated with three algorithmic costs,

- S – number of messages sent (network latency cost / synchronization cost),
- W – number of words of data moved (bandwidth cost / communication cost),
- F – number of local floating point operations performed (computational cost).

We describe our execution schedule model and show how S, W, and F are measured in any schedule in Section 2. Each quantity is accumulated along some path of dependent executions in the schedule. The sequence of executions done locally by any processor corresponds to one such path in the schedule, so our costs are at least as large as those incurred by any single processor during the execution of the schedule. The parallel execution time of the schedule is closely proportional to these three quantities, namely,

$$\max(S \cdot \alpha, W \cdot \beta, F \cdot \gamma) \leq \text{execution time} \leq S \cdot \alpha + W \cdot \beta + F \cdot \gamma.$$

Since our analysis will be asymptotic, we do not consider overlap between communication and computation and are able to measure the three quantities separately. Our model is similar to the LogP model [7] with $L = \alpha$, $o = g = \beta$, and no network capacity constraint.

Our theoretical analysis also precludes recomputation within a parallelization of an algorithm, as we associate an algorithm with a set of vertices, each of which is a computation, and assign them to unique processors. However, a parallel algorithm which employs recomputation has a different dependency graph structure, to which our analysis may subsequently be applied. While there are many existing parallel algorithms for the applications we explore that employ recomputation, to the best of our knowledge none of them perform less communication than ascribed by the recomputation-excluding lower bounds we present.

We reason about parallel algorithms by considering the dependency graphs of certain computations. We will first derive theoretical machinery for obtaining lower bounds on dependency graphs with certain expansion parameters and then show how this result yields

lower bounds on S, W, F for several algorithms in numerical linear algebra with common dependency structures. Most of our lower bounds apply to computations which have $\Omega(n^d)$ vertices, with a d-dimensional lattice dependency structure, and take the form

$$F \cdot S^{d-1} = \Omega\left(n^d\right), \quad W \cdot S^{d-2} = \Omega\left(n^{d-1}\right).$$

These bounds indicate that a growing amount of local computation, communication, and synchronization must be done to solve a larger global problem. Thus, the bounds are important because they highlight a scalability bottleneck dependent only on local processor/network speed and independent of the number of processors involved in the computation.

In particular, we show:

- For solving a dense n-by-n triangular system by substitution (TRSV),

$$F_{\mathrm{TRSV}} \cdot S_{\mathrm{TRSV}} = \Omega\left(n^2\right),$$

- For Gaussian elimination (GE) of a dense n-by-n matrix,

$$F_{\mathrm{GE}} \cdot S_{\mathrm{GE}}^2 = \Omega\left(n^3\right), \quad W_{\mathrm{GE}} \cdot S_{\mathrm{GE}} = \Omega\left(n^2\right),$$

- For computing an $(s+1)$-dimensional Krylov subspace basis with a $(2m+1)^d$-point stencil (defined in Section 6.3),

$$F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right), \quad W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right).$$

The lower bounds which we derive in this paper establish the communication optimality of the parallel algorithms for LU factorization given by Tiskin [16] and Solomonik and Demmel [20]. The parallel schedules for LU and QR in these papers are parameterized and exhibit a trade-off between synchronization and communication bandwidth cost. Our paper answers an open question posed by Tiskin at the end of [24], showing that it is not possible to achieve an optimal bandwidth cost for LU factorization without an associated increase in synchronization cost (within the limits of our assumptions).

In [20], a lower bound proof was given which demonstrated this trade-off for LU factorization. However, the proof argument in [20] did not consider the possibility of overlap between the communication necessary to factorize each block, and therefore was incorrect. This paper extends the idea of this tradeoff to a more general theoretical context, and presents a significantly corrected proof in this new framework. We show that Gaussian elimination is just one of many numerical algorithms whose dependency structure necessitates the tradeoff. In particular, we conjecture that our results extend to other dense matrix factorizations such as QR, problems in dynamic programming, and certain graph algorithms.

2 Previous work

Theoretical lower bounds on communication volume and synchronization can be parameterized by a local/fast memory of size M. Most previous work has considered the total sequential or parallel communication volume Q, which corresponds to the amount of data movement across the network (by all processors), or through the memory hierarchy. Hong and Kung [14] introduced sequential communication volume lower bounds for computations including n-by-n matrix multiplication, $Q_{\mathrm{MM}} = \Omega(n^3/\sqrt{M})$, the n-point FFT, $Q_{\mathrm{FFT}} = \Omega(n\log(n)/\log(M))$, and the d-dimensional diamond DAG (a Cartesian product of line graphs of length n), $Q_{\mathrm{dmd}} = \Omega(n^d/M^{1/(d-1)})$. Irony et al. [13] extended this approach to distributed-memory matrix multiplication on p processors, obtaining the bound $W_{\mathrm{MM}} = \Omega(n^3/p\sqrt{M})$. Aggarwal et al. [1] proved a version of the memory-independent lower bound $W_{\mathrm{MM}} = \Omega(n^3/p^{2/3})$, and Ballard et al. [2] explored the relationship between these memory-dependent and memory-independent lower bounds. Ballard et al. [3] extended the results for matrix multiplication to Gaussian elimination of n-by-n matrices and many other matrix algorithms with similar structure, finding

$$W_{\mathrm{GE}} = \Omega\left(\frac{n^3}{p\sqrt{\min(M, n^2/p^{2/3})}}\right).$$

Bender et al. [5] extended the sequential communication lower bounds introduced in [14] to sparse matrix vector multiplication. This lower bound is relevant to our analysis of Krylov subspace methods, which essentially perform repeated sparse matrix vector multiplications. However, [5] used a sequential memory hierarchy model and established bounds in terms of memory size and track (cacheline) size, while we focus on interprocessor communication.

Papadimitriou and Ullman [17] demonstrated tradeoffs for the 2-dimensional diamond DAG (a slight variant of that considered in [14]). They proved that the amount of computational work F_{dmd} along some execution path (in their terminology, execution time) is related to the communication volume Q_{dmd} and synchronization cost S_{dmd} as

$$F_{\mathrm{dmd}} \cdot Q_{\mathrm{dmd}} = \Omega\left(n^3\right) \text{ and } F_{\mathrm{dmd}} \cdot S_{\mathrm{dmd}} = \Omega\left(n^2\right).$$

These tradeoffs imply that in order to decrease the amount of computation done along the critical path of execution, more communication and synchronization must be performed. For instance, if an algorithm has 'execution time' cost of $F_{\mathrm{dmd}} = \Omega(nb)$, it requires $S_{\mathrm{dmd}} =$

$\Omega(n/b)$ synchronizations and a communication volume of $Q_{\text{dmd}} = \Omega(n^2/b)$. The tradeoff on $F_{\text{dmd}} \cdot S_{\text{dmd}}$ is a special case of the d-dimensional bubble latency lower bound tradeoff we derive in the next section, with $d = 2$. This diamond DAG tradeoff was also demonstrated by Tiskin [22].

Bampis et al. [4] considered finding the optimal schedule (and number of processors) for computing d-dimensional grid graphs, similar in structure to those we consider in Section 6. Their work was motivated by [17] and took into account dependency graph structure and communication, modeling the cost of sending a word between processors as equal to the cost of a computation.

We will introduce lower bounds that relate synchronization to computation and data movement along dependency paths. Our work is most similar to the approach in [17]; however, we attain bounds on W (the parallel communication volume along some dependency path), rather than Q (the total communication volume). Our theory obtains tradeoff lower bounds for a more general set of dependency graphs which allows us to develop lower bounds for a wider set of computations.

3 Theoretical model

We first introduce of the mathematical notation we will employ throughout this and later sections

- Sets are defined as uppercase letters (S, V).
- Vectors are defined as lowercase boldface letters ($\mathbf{v}$) and the ith element of $\mathbf{v}$ is indexed as v_i.
- Accordingly, matrices and tensors are defined as uppercase boldface letters ($\mathbf{A}, \mathbf{T}$), with elements A_{ij}, T_{ijk}.
- Functions and maps will be defined as Greek letters ($\sigma(n)$).
- Open/closed intervals in $\mathbb{R}$ are denoted (a, b) and $[a, b]$.

The **dependency graph** of a program is a directed acyclic graph (DAG) $G = (V, E)$. The vertices $V = I \cup Z \cup O$ correspond to either input values I (the vertices with in-degree 0), or the results of (distinct) operations, in which case they are either temporary (or intermediate) values Z, or outputs O. There is an edge $(u, v) \in E \subset V \times (Z \cup O)$ for each operand u of each operation v. These edges represent data dependencies, and impose limits on the parallelism available within the computation. For instance, if $G = (V, E)$ is a line graph with $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\}$, the computation is entirely sequential, and a lower bound on the execution time is the time it takes a single processor to compute $F = n - 1$ operations. Using graph expansion and hypergraph analysis, we will derive lower bounds for computation and communication for dependency graphs with certain properties. Then we will show that these lower bounds can be applied to several important problems in numerical linear algebra. These lower bounds have the form of tradeoffs between data transfer and synchronization cost and between work and synchronization cost, and form a conceptual communication wall which limits parallelism.

A **parallelization** of an algorithm corresponds to a coloring of its dependency graph, i.e., a partition of the vertices into p disjoint sets $V = \bigcup_{i=1}^{p} C_i$, where processor i for $i \in \{1, \ldots, p\}$ computes $C_i \cap (Z \cup O)$. We require that in any parallel execution among p processors, at least two processors compute $\lfloor |Z \cup O|/p \rfloor$ elements; this assumption is necessary to avoid the case of a single processor computing the whole problem sequentially (without parallel communication). Any vertex v of color i ($v \in C_i$) must be communicated to a different processor j if there is an edge from v to a vertex in C_j (though there need not necessarily be a message going directly between processor i and j, as the data can move through intermediate processors). We define each processor's **communicated set** as

$$T_i = \{u : (u, w) \in [(C_i \times (V \setminus C_i)) \cup ((V \setminus C_i) \times C_i)] \cap E\}.$$

We note that each T_i is a vertex separator in G between $C_i \setminus T_i$ and $V \setminus (C_i \cup T_i)$. For each processor i, a **communication schedule**, $\{m_i, \{R_{ij}\}, \{F_{ij}\}, \{M_{ij}\}\}$, defines a sequence of m_i time-steps. The time-steps may differ from processor to processor and relate to each other only via explicit communication in the schedule. At time-step $j \in \{1, \ldots, m_i\}$, processor i receives a set of values $R_{ij} \subset V$ (which are packed in one message and originate from a single processor), performs a computation (or no computation) $F_{ij} \subset V$, $|F_{ij}| \leq 1$, and sends a set of values $M_{ij} \subset V$ (which are also packed in one message and have a single destination processor). If both R_{ij} and M_{ij} are empty, no communication or synchronization is done by processor i at time-step j. Similarly, if $F_{ij} = \emptyset$, no computation is done by processor i at time-step j. We require that communication is point-to-point, so each message M_{ij} must be received on some unique processor k at some time-step l, that is $R_{kl} = M_{ij}$. The schedule must perform all computations $C_i = \bigcup_{j=1}^{m_i} F_{ij}$ and the messages sent and received by each processor i must include their communicated set $T_i \subset \bigcup_{j=1}^{m_i} M_{ij} \cup R_{ij}$ (a processor may send and receive values not in its communicated set in order to avoid synchronization of other processors). Further, the schedule should respect dependencies, that is, for each $f \in F_{ij}$ and each $(u, f) \in E$, if $u \in C_i$ then $u \in F_{ik}$ for some $k \in \{1, \ldots, j-1\}$, and if $u \notin C_i$, we require that $u \in R_{ik}$ for some $k \in \{1, \ldots, j\}$. Any value sent, $\forall u \in M_{ij}$, must either be an input ($u \in C_i \cap I$) or must have been previously received or computed by that process, which means there exists $k \in \{1, \ldots, j\}$ such that $u \in F_{ik}$ or $u \in R_{ik}$.

An execution of a parallel algorithm is associated with a communication schedule which can be represented by the weighted DAG $\bar{G} = (\bar{V}, \bar{E})$. For each processor $i \in \{1, \ldots, p\}$ and time-step $j \in \{1, \ldots, m_i\}$, we include vertex $(i, j) \in \bar{V}$. The graph includes 0-weighted edges $((i, k), (i, k+1), 0) \in \bar{E}$ for $k \in \{1, \ldots, m_i - 1\}$ (corresponding to sequentially executed computations F_{ik} and $F_{i,k+1}$). The graph also includes unit-weighted edges, $((i, j), (k, l), 1) \in \bar{E}$, for each nonempty message pair, $\emptyset \neq M_{ij} = R_{kl}$, where $i, k \in \{1, \ldots, p\}$, $i \neq k$, $j \in \{1, \ldots, m_i\}$, $l \in \{1, \ldots, m_k\}$.

We measure the execution costs by considering all **execution paths** Π that exist in the communication schedule DAG $\bar{G}$: each **execution path** $\pi \in \Pi$ is of the form $\pi = \{(i_1, j_1), \ldots, (i_n, j_n)\}$ where for each $k \in \{1, \ldots, n-1\}$, $((i_k, j_k), (i_{k+1}, j_{k+1}), w) \in \bar{E}$ and $w \in \{0, 1\}$. Since any two computations on π happen either on the same processor or follow a series of messages, the time to execute a path is a lower bound on the total execution time. The **computation cost** corresponds to the longest unweighted path through the schedule, i.e.,

$$F = \max_{\pi \in \Pi} \sum_{(i,j) \in \pi} |F_{ij}|.$$

Note that this value is greater than or equal to the work done by any single processor, i.e., for all i, $F \geq \sum_j |F_{ij}| = |C_i|$. The **bandwidth cost** of the communication schedule corresponds to the maximum vertex-weighted path through the schedule, with each vertex weighted by the size of the messages M_{ij} and R_{ij}, i.e.,

$$W = \max_{\pi \in \Pi} \sum_{(i,j) \in \pi} |M_{ij}| + |R_{ij}|.$$

Similarly, the bandwidth cost is at least as large as that incurred by any single processor, i.e., for all i, $W \geq \sum_j |M_{ij}| + |R_{ij}|$. Accordingly, the **latency cost** corresponds to the longest edge-weighted path through the schedule. Defining the edge-to-weight mapping $\hat{\omega}(u, v) = w$ for each $(u, v, w) \in \bar{E}$, the maximum weighted path in the schedule is given by

$$S = \max_{\{(i_1,j_1),\ldots,(i_n,j_n)\} \in \Pi} \sum_{k=1}^{n-1} \hat{\omega}\left((i_k, j_k), (i_{k+1}, j_{k+1})\right).$$

Our goal will be to lower bound the bandwidth and latency costs for any parallelization and communication schedule of a given dependency graph.

4 General lower bound theorem via bubble expansion

In this section, we introduce the concept of dependency bubbles and their expansion. Bubbles represent sets of interdependent computations, and their expansion allows us to analyze the cost of computation and communication for any parallelization and communication schedule. We will show that if a dependency graph has a path along which bubbles expand as some function of the length of the path, any parallelization of this dependency graph must sacrifice synchronization or incur higher computational and data volume costs, which scale with the total size and cross-section size (minimum cut size) of the bubbles, respectively.

4.1 Bubble expansion

Given a dependency graph $G = (V, E)$, we say $v_n \in V$ **depends on** $v_1 \in V$ if and only if there is a path $\mathcal{P} \subset V$ connecting v_1 to v_n, i.e., $\mathcal{P} = \{v_1, \ldots, v_n\}$ such that $\{(v_1, v_2), \ldots, (v_{n-1}, v_n)\} \subset E$. We denote a sequence of (not necessarily adjacent) vertices $\{w_1, \ldots, w_n\}$ a **dependency path**, if for $i \in \{1, \ldots, n-1\}$, w_{i+1} is dependent on w_i.

The (**dependency**) **bubble** around a dependency path $\mathcal{P}$ connecting v_1 to v_n is a sub-DAG $\beta(G, \mathcal{P}) = (V_\beta, E_\beta)$ where $V_\beta \subset V$, each vertex $u \in V_\beta$ lies on a dependency path $\{v_1, \ldots, u, \ldots, v_n\}$ in G, and $E_\beta = (V_\beta \times V_\beta) \cap E$. This bubble corresponds to all vertices which must be computed between the start and end of the path. Equivalently, the bubble may be defined as the union of all paths between v_1 and v_n.

4.2 Lower bounds based on bubble expansion

A $\frac{1}{q}$**-balanced vertex separator** $Q \subset V$ of a graph $G = (V, E)$ splits $V \setminus Q = V_1 \cup V_2$ so that $\min(|V_1|, |V_2|) \geq \lfloor |V|/q \rfloor$ and $E \subset (V_1 \times V_1) \cup (V_2 \times V_2) \cup (Q \times V) \cup (V \times Q)$. We denote the minimum size $|Q|$ of a $\frac{1}{q}$-balanced separator Q of G as $\chi_q(G)$. If $\beta(G, \mathcal{P})$ is the dependency bubble formed around path $\mathcal{P}$, we say $\chi_q(\beta(G, \mathcal{P}))$ is its **cross-section expansion**.

Definition 4.1 We call a directed-acyclic graph G a (ϵ, σ)**-path-expander** if there exists a dependency path $\mathcal{P}$ in the graph G and a positive integer constant $k \ll |\mathcal{P}|$ such that every subpath $\mathcal{R} \subset \mathcal{P}$ of length $|\mathcal{R}| \geq k$ has bubble $\beta(G, \mathcal{R}) = (V_\beta, E_\beta)$ with cross-section expansion $\chi_q(\beta(G, \mathcal{R})) = \Omega(\epsilon(|\mathcal{R}|))$ for any real constant $q \ll |\mathcal{P}|$, $q > 2$, and bubble size $|V_\beta| = \Theta(\sigma(|\mathcal{R}|))$, for some real-valued functions ϵ, σ which for real numbers $b \geq k$ are positive and increasing with $1 \leq \epsilon(b+1) - \epsilon(b) \leq c_\epsilon \epsilon(b)$ and $1 \leq \sigma(b+1) - \sigma(b) \leq c_\sigma \sigma(b)$ for some positive real constants $c_\epsilon, c_\sigma \ll |\mathcal{P}|$.

Theorem 4.1 (General bubble lower bounds)**.** *Suppose a dependency graph G is a (ϵ, σ)-path-expander about dependency path $\mathcal{P}$. Then, for any parallelization of G in which no processor computes more than half of the vertices of $\beta(G, \mathcal{P})$, and for any communication schedule, there exists an integer $b \in [k, |\mathcal{P}|]$ such that the computation (F), bandwidth (W), and latency (S) costs incurred are*

$$F = \Omega\left(\sigma(b) \cdot |\mathcal{P}|/b\right), \qquad W = \Omega\left(\epsilon(b) \cdot |\mathcal{P}|/b\right), \qquad S = \Omega\left(|\mathcal{P}|/b\right).$$

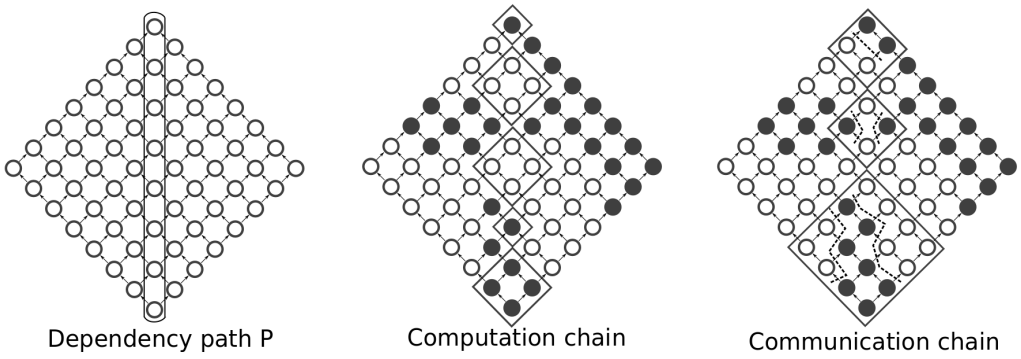


Figure 1: Illustration of the construction in the proof of Theorem 4.1 in the case of a diamond DAG (e.g., [17]), depicting a dependency path, and communication and computation chains about that path, for a 2-processor parallelization.

Proof. We consider any possible parallelization, which implies a coloring of the vertices of $G = (V, E)$, $V = \bigcup_{i=1}^{p} C_i$, and show that any communication schedule $\{m_i, \{R_{ij}\}, \{F_{ij}\}, \{M_{ij}\}\}$ for $i \in \{1, \ldots, p\}, j \in \{1 \ldots, m_i\}$, corresponding to the communication schedule graph $\bar{G} = (\bar{V}, \bar{E})$, as defined in Section 3, incurs the desired computation, bandwidth, and latency costs. Our proof technique works by defining a chain of bubbles within G in a way that allows us to accumulate the costs along the chain.

The tradeoff between work and synchronization, $F = \Omega(\sigma(b) \cdot |\mathcal{P}|/b)$ and $S = \Omega(|\mathcal{P}|/b)$, can be derived by considering a computation chain: a sequence of monochrome bubbles along $\mathcal{P}$, each corresponding to a set of computations performed sequentially by some processor (see computation chain in Figure 1). However, to obtain the bandwidth lower bound, we must instead show that there exists a sequence of bubbles in which some processor computes a constant fraction of each bubble; we then sum the bandwidth costs incurred by each bubble in the sequence. We show a communication chain (a sequence of multicolored bubbles) for a diamond DAG in Figure 1.

Every subpath $\mathcal{R} \subset \mathcal{P}$ of length $|\mathcal{R}| \geq k$ induces a bubble $\beta(G, \mathcal{R}) = (V_\beta, E_\beta)$ of size $|V_\beta| = \Theta(\sigma(|\mathcal{R}|))$. The Θ-notation implies that there exists a positive finite constant $d \ll |\mathcal{P}|$ such that for all $|\mathcal{R}| \geq k$,

$$\sigma(|\mathcal{R}|)/d \leq |V_\beta| \leq d \cdot \sigma(|\mathcal{R}|).$$

We also fix the constant

$$q = \max\left\{d^2(c_\sigma + 1) + 1, d \cdot \sigma(k)\right\};$$

note that $q \ll |\mathcal{P}|$, since $k, d, c_\sigma, \sigma(k) \ll |\mathcal{P}|$.

We define the bubbles via the following procedure, which partitions the path $\mathcal{P}$ into subpaths by iteratively removing leading subpaths. Assume we have defined subpaths $\mathcal{R}_j$ for $j \in \{1, \ldots, i-1\}$. Let the tail (remaining trailing subpath) of the original path be

$$\mathcal{T}_i = \mathcal{P} \setminus \bigcup_{j=1}^{i-1} \mathcal{R}_j = \left\{t_1, \ldots, t_{|\mathcal{T}_i|}\right\}.$$

Our procedure defines the next leading subpath of length r, $\mathcal{R}_i = \{t_1, \ldots, t_r\}, k \leq r \leq |\mathcal{T}_i|$, with bubble $\beta(G, \mathcal{R}_i) = (V_i, E_i)$. Suppose processor l computes t_1. The procedure picks the shortest leading subpath of $\mathcal{T}_i$ of length $r \geq k$ which satisfies the following two conditions, and terminates if no such path can be defined.

Condition 1: The subset of the bubble that processor l computes, $C_l \cap V_i$, is of size at least $|C_l \cap V_i| \geq \lfloor |V_i|/q \rfloor$

Condition 2: The subset of the bubble that processor l does not compute, $V_i \setminus C_l$, is also of size at least $|V_i \setminus C_l| \geq \lfloor |V_i|/q \rfloor$

Let c be the number of subpaths the procedure outputs for the given parallelization of G and $\mathcal{T} = \mathcal{P} \setminus \bigcup_{j=1}^{c} \mathcal{R}_j = \{t_1, \ldots, t_{|\mathcal{T}|}\}$ be the tail remaining at the end of the procedure. We consider the two cases, $|\mathcal{T}| = \Omega(|\mathcal{P}|)$ and $\sum_j |\mathcal{R}_j| = \Omega(|\mathcal{P}|)$, at least one of which must hold. We show that in either case, the theorem is true for some value of b.

If $|\mathcal{T}| = \Omega(|\mathcal{P}|)$ (the tail is long), we show that Condition 1 must be satisfied for any leading subpath of $\mathcal{T}$. Our proof works by induction on the length of the leading subpath $k \leq r \leq |\mathcal{T}|$, with the subpath given by $\mathcal{K}_r = \{t_1, \ldots, t_r\}$. We define the bubble about $\mathcal{K}_r$ as $\beta(G, \mathcal{K}_r) = (V_r, E_r)$. When $r = k$, Condition 1 is satisfied because $|V_r|/q \leq d \cdot \sigma(k)/q \leq 1$ and processor l computes at least one element, t_1.

For $r > k$, we have

$$|V_r| \leq d \cdot \sigma(r) \leq d(c_\sigma + 1) \cdot \sigma(r-1) \leq \frac{q-1}{d} \cdot \sigma(r-1).$$

Further, by induction, Condition 1 was satisfied for $\mathcal{K}_{r-1}$ which implies Condition 2 was not satisfied for $\mathcal{K}_{r-1}$ (otherwise the procedure would have terminated with a path of length $r-1$). Now, using bounds on bubble growth we that since Condition 2 was not satisfied for

$\mathcal{K}_{r-1}$, Condition 1 has to be satisfied for the subsequent bubble, $\mathcal{K}_r$,

$$|C_l \cap V_r| \geq |C_l \cap V_{r-1}| \geq |V_{r-1}| - \left\lfloor \frac{|V_{r-1}|}{q} \right\rfloor$$
$$\geq \frac{q-1}{q}|V_{r-1}| \geq \frac{q-1}{dq}\sigma(r-1) \geq \frac{1}{q}|V_r| \geq \left\lfloor \frac{|V_r|}{q} \right\rfloor,$$

so Condition 1 holds for $\mathcal{K}_r$ for $r \in \{k, \ldots, |\mathcal{T}|\}$. Further, since the tail is long, $|\mathcal{T}| = \Omega(|\mathcal{P}|)$, due to Condition 1, processor l must compute

$$F \geq \left\lfloor |V_{\beta(G,\mathcal{T})}|/q \right\rfloor = \Omega(\sigma(|\mathcal{T}|))$$

vertices. Since, by assumption, no processor can compute more than half of the vertices of $\beta(G, \mathcal{P})$, we claim there exists a subpath $\mathcal{Q}$ of $\mathcal{P}$, $\mathcal{T} \subset \mathcal{Q} \subset \mathcal{P}$, where processor l computes $\left\lfloor |V_{\beta(G,\mathcal{Q})}|/q \right\rfloor$ vertices and does not compute $\left\lfloor |V_{\beta(G,\mathcal{Q})}|/q \right\rfloor$ vertices. The path $\mathcal{Q}$ may always be found to satisfy these two conditions simultaneously, since we can grow $\mathcal{Q}$ backward from $\mathcal{T}$ until Condition 2 is satisfied, i.e., processor l does not compute at least $\lfloor |V_{\beta(G,\mathcal{Q})}|/q \rfloor$ vertices, and we will not violate the first condition that $(|C_l \cap V_{\beta(G,\mathcal{Q})}| \geq \lfloor |V_{\beta(G,\mathcal{Q})}|/q \rfloor$, which holds for $\mathcal{T}$, due to bounds on growth of $|V_{\beta(G,\mathcal{Q})}|$. The proof of this assertion is the same as the inductive proof above which showed that Condition 1 holds on $\mathcal{K}_r$. So, processor l must incur a communication cost proportional to a $\frac{1}{q}$-balanced vertex separator of $\beta(G, \mathcal{Q})$ with size $W = \Omega(\epsilon(|\mathcal{Q}|)) = \Omega(\epsilon(|\mathcal{T}|))$. Since these costs are incurred along a path in the schedule consisting of the work and communication done only by processor l, the bounds hold for $b = |\mathcal{T}|$ (note that $\Omega(|\mathcal{P}|/|\mathcal{T}|) = \Omega(1)$, because $|\mathcal{T}| = \Omega(|\mathcal{P}|)$).

In the second case, $\sum_j |\mathcal{R}_j| = \Omega(|\mathcal{P}|)$, the procedure generates subpaths with a total size proportional to the size of $\mathcal{P}$. For each $i \in \{1, \ldots, c\}$, consider the time-step m during which processor l computed the first vertex t_1 on the path $\mathcal{R}_i$, that is, $(l, m) \in \bar{V}$ where $l \in \{1, \ldots, p\}$ and $m \in \{1 \ldots m_l\}$, such that $F_{lm} = \{t_1\}$ (recall that each process computes at most one vertex every time-step). We choose the smallest $s \geq m$ such that $C_l \cap V_i \subset \bigcup_{m'=m}^{s} F_{lm'}$ (processor l computes its part of V_i between time-steps m and s). Now consider the time-step v on processor u, $(u, v) \in \bar{V}$, during which the last vertex t_r on the path $\mathcal{R}_j$ was computed, that is, $(u, v) \in \bar{V}$ where $u \in \{1, \ldots, p\}$ and $v \in \{1, \ldots, m_u\}$, such that $F_{uv} = \{t_r\}$. Note that for some $z \in V_i$, $F_{ls} = \{z\}$, (otherwise, s can be taken to be to $s-1$, and so is not the smallest) and t_r is dependent on z (t_r is dependent on all vertices in the bubble). So, there will be an execution path $\pi_i = \{(l, m), \ldots, (l, s), \ldots, (u, v)\} \subset \bar{V}$ in the communication schedule. This path has outgoing messages $\{M_{lm}, \ldots, M_{ls}, \ldots, M_{uv}\}$ and incoming messages $\{R_{lm}, \ldots, R_{ls}, \ldots, R_{uv}\}$ that include all vertices in V_i which processor l must communicate to compute $V_i \cap C_l$, which is given by the set

$$\hat{T}_{il} = \{u : (u, w) \in [(C_l \times (V_i \setminus C_l)) \cup ((V_i \setminus C_l) \times C_l)] \cap E_i\},$$

which is a separator of $\beta(G, \mathcal{R}_i)$ and is $\frac{1}{q}$-balanced due to Conditions 1 and 2 in the definition of $\mathcal{R}_i$. We use the lower bound on the minimum separator of a bubble to obtain a lower bound on the size of the communicated set for processor l in the ith bubble,

$$|\hat{T}_{il}| \geq \chi_q(\beta(G, \mathcal{R}_i)) = \Omega\left(\epsilon(|\mathcal{R}_i|)\right),$$

where we are able to bound the growth of $\beta(G, \mathcal{R}_i)$, since $|\mathcal{R}_i| \geq k$. There exists a dependency path between the last element of $\mathcal{R}_i$ and the first of $\mathcal{R}_{i+1}$ since they are subpaths of $\mathcal{P}$, so every bubble $\beta(G, \mathcal{R}_i)$ must be computed entirely before any members of $\beta(G, \mathcal{R}_{i+1})$ are computed. Therefore, there is an execution path $\pi_{\text{critical}} \subset \bar{V}$ in the communication schedule which contains $\pi_i \subset \pi_{\text{critical}}$ as a subpath for every $i \in \{1, \ldots, c\}$. Communication and computation along π_{critical} can be bounded below by

$$F = \sum_{(i,j) \in \pi_{\text{critical}}} |F_{ij}| \geq \sum_{i=1}^{c} \frac{1}{q}|\beta(G, \mathcal{R}_i)| = \Omega\left(\sum_{i=1}^{c} \sigma(|\mathcal{R}_i|)\right),$$

$$W = \sum_{(i,j) \in \pi_{\text{critical}}} |M_{ij}| + |R_{ij}| \geq \sum_{i=1}^{c} \chi_q(\beta(G, \mathcal{R}_i)) = \Omega\left(\sum_{i=1}^{c} \epsilon(|\mathcal{R}_i|)\right).$$

Further, since each bubble contains vertices computed by multiple processes, between the first and last vertex on the subpath forming each bubble, a network latency cost of at least one must be incurred per bubble, therefore,

$$S \geq = \Omega(c).$$

Because $\sigma(b+1) - \sigma(b) \geq 1$ for $b \geq k \geq |\mathcal{R}_i|$, and the sum of all the lengths of the subpaths is bounded ($\sum_i |\mathcal{R}_i| \leq |\mathcal{P}|$), the above lower bounds for F and W are minimized when all $\mathcal{R}_i$ are of the same length[1]. Picking this length as b, that is $|\mathcal{R}_i| = b = \Theta(|\mathcal{P}|/c)$ for all i, leads to a simplified form for the bounds,

$$F = \Omega\left(\sigma(b) \cdot |\mathcal{P}|/b\right), \quad W = \Omega\left(\epsilon(b) \cdot |\mathcal{P}|/b\right), \quad S = \Omega\left(|\mathcal{P}|/b\right).$$

□

[1]This mathematical relation can be demonstrated by a basic application of Lagrange multipliers.

Corollary 4.2 (d-dimensional bubble lower bounds)**.** *Let $\mathcal{P}$ be a dependency path in G, such that every subpath $\mathcal{R} \subset \mathcal{P}$ of length $|\mathcal{R}| \geq k$, where $k \ll |\mathcal{P}|$, has bubble $\beta(G, \mathcal{R}) = (V_\beta, E_\beta)$ with cross-section expansion $\chi_q(\beta(G, \mathcal{R})) = \Omega(|\mathcal{R}|^{d-1})$ for any constant $2 \leq q \ll |\mathcal{P}|$ and bubble size $|V_\beta| = \Theta(|\mathcal{R}|^d)$, for an integer $2 \leq d \ll k$. The computation, bandwidth, and latency costs incurred by any parallelization of G in which no processor computes more than half of the vertices of $\beta(G, \mathcal{P})$, and with any communication schedule, must obey the relations*

$$F \cdot S^{d-1} = \Omega\left(|\mathcal{P}|^d\right), \quad W \cdot S^{d-2} = \Omega\left(|\mathcal{P}|^{d-1}\right).$$

Proof. This is an application of Theorem 4.1 with $\epsilon(b) = b^{d-1}$ and $\sigma(b) = b^d$. The theorem yields

$$F = \Omega\left(b^{d-1} \cdot |\mathcal{P}|\right), \quad W = \Omega\left(b^{d-2} \cdot |\mathcal{P}|\right), \quad S = \Omega\left(|\mathcal{P}|/b\right).$$

These equations can be manipulated algebraically to obtain

$$F \cdot S^{d-1} = \Omega\left(|\mathcal{P}|^d\right), \quad W \cdot S^{d-2} = \Omega\left(|\mathcal{P}|^{d-1}\right).$$

□

5 Lower bounds on lattice hypergraph cuts

For any hypergraph $H = (V, E)$, we say a hyperedge $e \in E$ is **internal** to some $V' \subset V$ if $e \subset V'$. If no $e \in E$ is adjacent to (i.e., contains) a $v \in V' \subset V$, then say V' is **disconnected** from H. A $\frac{1}{q}$**-balanced** (**hyperedge**) **cut** of is a subset of E whose removal from H partitions $V = V_1 \cup V_2$ with $\min(|V_1|, |V_2|) \geq \frac{1}{q}|V|$ such that all remaining (uncut) hyperedges are internal to one of the two parts.

We define a d-dimensional **lattice hypergraph** $H = (V, E)$ of breadth n, with $|V| = \binom{n}{d}$ vertices and $|E| = \binom{n}{d-1}$ hyperedges. Each vertex is represented as $v_{i_1,\ldots,i_d} = (i_1, \ldots, i_d)$ for $\{i_1, \ldots, i_d\} \in \{1, \ldots, n\}^d$ with $i_1 < \cdots < i_d$. Each hyperedge connects all vertices which share $d-1$ indices, that is $e_{j_1,\ldots,j_{d-1}}$ for $\{j_1, \ldots, j_{d-1}\} \in \{1, \ldots, n\}^{d-1}$ with $j_1 < \cdots < j_{d-1}$ includes all vertices $v_{i_1,\ldots,i_d}$ for which $\{j_1, \ldots, j_{d-1}\} \subset \{i_1, \ldots, i_d\}$. There are $n-(d-1)$ vertices per hyperedge, and each vertex appears in d hyperedges. Each hyperedge intersects $(d-1)(n-(d-1))$ other hyperedges, each at a unique vertex.

A key step in the lower bound proofs in [13] and [3] was the use of an inequality introduced by Loomis and Whitney [15]. We will use this inequality (in the following form) to prove a lower bound on the cut size of a lattice hypergraph.

Theorem 5.1 (Loomis-Whitney)**.** *Let V be a set of d-tuples $(i_1, \ldots, i_d) \in \mathbb{N}^d$, and consider projections $\pi_j \colon \mathbb{N}^d \to \mathbb{N}^{d-1}$ for $j \in \{1, \ldots, d\}$ defined as*

$$\pi_j(i_1, \ldots, i_d) = (i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_d),$$

then the cardinality of V is bounded by,

$$|V| \leq \prod_{j=1}^{d} |\pi_j(V)|^{1/(d-1)}.$$

Theorem 5.2. *For $2 \leq d, q \ll n$, the minimum $\frac{1}{q}$-balanced cut of a d-dimensional lattice hypergraph $H = (V, E)$ is of size $\epsilon_q(H) = \Omega(n^{d-1}/q^{(d-1)/d})$.*

Proof. We prove Theorem 5.2 by induction on the dimension, d. In the base case $d = 2$, we must show that $\epsilon_q(H) = \Omega(n/\sqrt{q})$. Consider any $\frac{1}{q}$-balanced cut $Q \subset E$, which splits the vertices into two disjoint sets V_1 and V_2. Note that in 2 dimensions, every pair of hyperedges overlaps, i.e., for $i_1, i_2 \in \{1, \ldots, n\}$ with $i_1 < i_2$, $e_{i_1} \cap e_{i_2} = \{v_{i_1,i_2}\}$. If the first partition, V_1, has an internal hyperedge, then since every pair of hyperedges overlaps, V_1 is adjacent to all hyperedges in H. Therefore, every hyperedge adjacent to the other part $V_2 \subset V$ must be in the cut, Q. On the other hand, if V_1 has no internal hyperedges, then all hyperedges adjacent to V_1 must connect both parts, and thus are cut. So, without loss of generality we will assume that V_2 is disconnected after the cut. We now argue that $\Omega(n/\sqrt{q})$ hyperedges must be cut to disconnect V_2.

Since the cut is $\frac{1}{q}$-balanced, we know that $|V_2| \geq n(n-1)/(2q)$. To disconnect each $v_{i_1,i_2} \in V_2$, both adjacent hyperedges (e_{i_1} and e_{i_2}) must be cut. We can bound from below the cut size by first obtaining a lower bound on the product of the sizes of the projections $\pi_1(v_{i_1,i_2}) = i_2$ and $\pi_2(v_{i_1,i_2}) = i_1$ via the Loomis-Whitney inequality (Theorem 5.1),

$$|\pi_1(V_2)| \cdot |\pi_2(V_2)| \geq |V_2| \geq n(n-1)/(2q),$$

and then concluding

$$\begin{aligned}
\epsilon_q(H) &= |\pi_1(V_2) \cup \pi_2(V_2)| \\
\epsilon_q(H) &\geq \frac{1}{2}\left(|\pi_1(V_2)| + |\pi_2(V_2)|\right) \\
&\geq \sqrt{|\pi_1(V_2)| \cdot |\pi_2(V_2)|} \geq \sqrt{n(n-1)/(2q)} \\
&= \Omega\left(n/\sqrt{q}\right),
\end{aligned}$$

since the size of the union of the two projections equals the number of hyperedges that must be cut to disconnect V_2.

For the inductive step, we assume that the theorem holds for dimension $d-1$ and prove that it must also hold for dimension d, where $d \geq 3$. In d dimensions, we define a **hyperplane** $x_{k_1,\ldots,k_{d-2}}$ for each $\{k_1,\ldots,k_{d-2}\} \in \{1,\ldots,n\}^{d-2}$ with $k_1 < \cdots < k_{d-2}$ as the set of all hyperedges $e_{j_1,\ldots,j_{d-1}}$ which satisfy $\{k_1,\ldots,k_{d-2}\} \subset \{j_1,\ldots,j_{d-1}\}$. Thus, each of the $|X| = \binom{n}{d-2}$ hyperplanes contains $n-(d-2)$ hyperedges, and each hyperedge is in $d-1$ hyperplanes. Note that each hyperplane shares a unique hyperedge with $(d-2)(n-(d-2))$ other hyperplanes. Further, each hyperedge in a hyperplane intersects each other hyperedge in the same hyperplane in a unique vertex, and the set of all these vertices are precisely those sharing the $d-2$ indices defining the hyperplane.

Consider any $\frac{1}{q}$-balanced hypergraph edge cut $Q \subset E$. Since all hyperedges which contain vertices in both V_1 and V_2 must be part of the cut Q, all vertices are either disconnected completely by the cut or remain in hyperedges which are all internal to either V_1 or V_2. Let $U_1 \subset V_1$ be the vertices contained in a hyperedge internal to V_1 and let $U_2 \subset V_2$ be the vertices contained in a hyperedge internal to V_2. Since both V_1 and V_2 contain $\lfloor n^d/q \rfloor$ vertices, either $\lfloor n^d/2q \rfloor$ vertices must be in internal hyperedges within both V_1 as well as V_2, that is,

$$\text{case (i): } |U_1| \geq \lfloor n^d/2q \rfloor \text{ and } |U_2| \geq \lfloor n^d/2q \rfloor,$$

or there must be $\lfloor n^d/2q \rfloor$ vertices that are disconnected completely by the cut,

$$\text{case (ii): } |(V_1 \setminus U_1) \cup (V_2 \setminus U_2)| \geq \lfloor n^d/2q \rfloor.$$

In case (i), since both U_1 and U_2 have at least $\lfloor n^d/2q \rfloor$ vertices, we know that there are at least $|U_1|/(n-(d-1)) \geq \lfloor n^{d-1}/2q \rfloor$ hypergraph edges W_1 which are internal to V_1 after the cut Q, and a similar set of hyperedges W_2 internal to V_2. We now obtain a lower bound on the size of the cut Q for this case by counting the hyperplanes which Q must contain. Our argument relies on the idea that if two hypergraph edges are in the same hyperplane, the entire hyperplane must be disconnected (all of its hyperedges must be part of the cut Q) in order to disconnect the two edges. This allows us to bound the number of hyperplanes which must be disconnected in order for W_1 to be disconnected from W_2.

We define a new $(d-1)$-dimensional lattice hypergraph $H' = (E, X)$, with vertices and hyperedges equal to the hyperedges and hyperplanes of the original hypergraph H. The cut Q induces a $\frac{1}{2q}$-balanced cut on H' since it creates two disconnected partitions of hyperedges: W_1 and W_2, each of size $\lfloor n^{d-1}/2q \rfloor$. We can assert a lower bound on the size of any $\frac{1}{2q}$-balanced cut of H' by induction,

$$\epsilon_q(H') = \Omega\left(n^{d-2}/(2q)^{(d-2)/(d-1)}\right) = \Omega\left(n^{d-2}/q^{(d-2)/(d-1)}\right).$$

This lower bound on cut size of H' yields a lower bound on the number of hyperplanes which must be cut to disconnect the hyperedges into two balanced sets W_1 and W_2. Remembering that disconnecting each hyperplane requires cutting all of its internal $n-(d-2)$ hyperedges (and also that each pair of hyperplanes overlap on at most one hyperedge), allows us to conclude that the number of hyperedges cut (in Q) must be at least

$$\epsilon_q(H) \geq \frac{(n-(d-2))}{d-1}\epsilon_q(H') = \Omega\left(n^{d-1}/q^{(d-2)/(d-1)}\right).$$

The quantity on the right is always larger than the lower bound we are trying to prove, $\epsilon_q(H) = \Omega(n^{d-1}/q^{(d-1)/d})$, so the proof for this case is complete.

In case (ii), we know that $\lfloor n^d/2q \rfloor$ vertices $\bar{U} \subset V$ are disconnected by the cut (before the cut, every vertex was adjacent to d hyperedges). We define d projections,

$$\pi_j(v_{i_1,\ldots,i_d}) = (i_1,\ldots,i_{j-1},i_{j+1},\ldots,i_d)$$

for $j \in \{1,\ldots,d\}$ corresponding to each of d hyperedges adjacent to $v_{i_1\ldots i_d}$. We apply the Loomis-Whitney inequality (Theorem 5.1) to obtain a lower bound on the product of the size of the projections,

$$\prod_{j=1}^{d} |\pi_j(\bar{U})|^{1/(d-1)} \geq |\bar{U}| \geq \lfloor n^d/2q \rfloor,$$

and then conclude with a lower bound on the number of hyperedges in the cut of H,

$$\begin{aligned}\epsilon_q(H) &\geq |\bigcup_{j=1}^{d} \pi_j(\bar{U})| \geq \frac{1}{d}\sum_{j=1}^{d} |\pi_j(\bar{U})| \geq \frac{1}{d}\prod_{j=1}^{d} |\pi_j(\bar{U})|^{1/d} \\ &= \Omega\left(\left(n^d/2q\right)^{(d-1)/d}\right) = \Omega\left(n^{d-1}/q^{(d-1)/d}\right),\end{aligned}$$

where we discard the constant $\frac{1}{d}$ by applying our assumption that $d \ll n$. By induction, this lower bound holds for all $2 \leq d \ll n$. $\square$

6 Applications

In this section, we apply the general theorems derived in the previous sections to obtain lower bounds on the costs associated with a few specific numerical linear algebra algorithms. We treat the dependency graphs of the algorithms in a general manner by reducing them to lattice hypergraphs.

Consider a dependency graph $G = (V, E)$ and a partition $\{E_i\}$ of its edge set E. Let V_i be all the vertices adjacent to the edge partition E_i for each i (while $\{E_i\}$ is a disjoint partition of E, $\{V_i\}$ is not necessarily a disjoint partition of V). We can define a hypergraph $H = (V, D)$ based on G, where in each hyperedge $d_i \in D$, every pair of vertices $u, v \in d_i$ is connected in G via a path consisting of edges in E_i. By defining H in this manner, any cut of $C \subset E$ corresponds to a hypergraph cut of H with at most $|C|$ hyperedges, which may be obtained by cutting all hyperedges $d_i \in D$ corresponding to parts E_i which contain cut edges, i.e., $\exists e \in E_i$ such that $e \in C$.

We will also employ hyperedges to obtain lower bounds on sets of vertices connected via arbitrary reduction (sum) trees. Any reduction tree $T = (R, E)$ which sums a set of vertices $S \subset R$ must connect each pair of vertices in S. Therefore, we can define a hypergraph edge corresponding to this reduction tree, which contains the edges in S (ignoring the intermediate vertices $R \setminus S$ which depend on the particular tree), with the edge partition corresponding to the hyperedge being E for any possible reduction tree $T = (R, E)$.

6.1 Triangular solve

First, we consider a parameterized family of dependency graphs $G_{\mathrm{TRSV}}(n)$ associated with an algorithm for the triangular solve (TRSV) operation. In TRSV, we are interested in computing a vector $\mathbf{x}$ of length n, given a dense nonsingular lower-triangular matrix $\mathbf{L}$ and a vector $\mathbf{y}$, satisfying

$$\mathbf{L} \cdot \mathbf{x} = \mathbf{y},$$

i.e., $\sum_{j=1}^{i} L_{ij} \cdot x_j = y_i$, for $i \in \{1, \ldots, n\}$. A sequential TRSV implementation is given in Algorithm 1. For convenience, we

Algorithm 1 Triangular solve (TRSV) algorithm

$\mathbf{x} = \mathrm{TRSV}(\mathbf{L}, \mathbf{y}, n)$

1 **for** $i = 1$ **to** n
2 **for** $j = 1$ **to** $i - 1$
3 $Z_{ij} = L_{ij} \cdot x_j$
4 $x_i = \left(y_i - \sum_{j=1}^{i-1} Z_{ij}\right) / L_{ii}$

introduced the intermediate matrix $\mathbf{Z}$ (which need not be formed explicitly in practice), and corresponding intermediate 'update' vertices $\{Z_{ij} : i, j \in \{1, \ldots, n\}, j < i\}$. We see that the computation of Z_{ij} for $i = \{2, \ldots, n\}$ and some $j < i$ depends on the computation of x_j, which in turn influences the computations of Z_{jk} for all $k < j$.

6.1.1 Lower bounds

For fixed n, alternative orders exist for the summation on line 4, leading to multiple dependency graphs $G_{\mathrm{TRSV}}(n)$. However, any order of this summation must eventually combine all partial sums; therefore, the vertices corresponding to the computation of each x_i, i.e., Z_{ij} for all $j \in \{1, \ldots, i-1\}$, must be connected via some reduction tree. We will define a 2-dimensional lattice hypergraph $H_{\mathrm{TRSV}} = (V_{\mathrm{TRSV}}, E_{\mathrm{TRSV}})$, which will allow us to obtain a communication lower bound for all possible orderings of this computation (i.e., all possible $G_{\mathrm{TRSV}}(n)$), in which we will omit the output and input vertices ($\mathbf{x}$ and $\mathbf{y}$),

$$\begin{aligned} V_{\mathrm{TRSV}} =& \{Z_{ij} : i, j \in \{1, \ldots, i-1\}, i > j\}, \\ E_{\mathrm{TRSV}} =& \{e_i : i \in \{1, \ldots, n\}\} \text{ where} \\ & e_i = \{Z_{ij} : j \in \{1, \ldots, i-1\}\} \cup \{Z_{ki} : k \in \{i+1, \ldots, n\}\}. \end{aligned}$$

The hyperedges E_{TRSV} can be enumerated with respect to the vector $\mathbf{x}$ or $\mathbf{y}$; the ith hyperedge $e_i \in E_{\mathrm{TRSV}}$ includes all intermediate values which are dependencies of x_i (Z_{ij} for $j \in \{1, \ldots, i-1\}$) or dependent on x_i (Z_{ki} for $k \in \{i+1, \ldots, n\}$). This hypergraph is depicted in Figure 2.

Lemma 6.1. *Any vertex separator of any dependency graph $G_{\mathrm{TRSV}}(n)$ which subdivides the $n(n-1)/2$ intermediate vertices $\mathbf{Z}$ into two disjoint sets of size $\lfloor n^2/2q \rfloor$ where $2 \leq q \ll n$, must have size at least*

$$\chi_q(G_{\mathrm{TRSV}}(n)) = \Omega\left(n/\sqrt{q}\right).$$

Proof. Consider any vertex separator S on $G_{\mathrm{TSRV}}(n)$ which subdivides $\mathbf{Z}$ into two sets of size $\lfloor n^2/2q \rfloor$. Any graph $G_{\mathrm{TRSV}}(n)$ may contain vertices corresponding to $\mathbf{x}$, $\mathbf{Z}$ or other intermediate vertices which are intermediate nodes in a reduction tree, whose sum

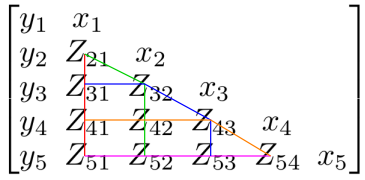


Figure 2: Depiction of the hypergraph H_{TRSV} along with the inputs and outputs; each line of a different color corresponds to a hyperedge.

contributes to x_i for some i (including vertices from $\mathbf{y}$ does not affect the connectivity of vertices in $\mathbf{Z}$). We now show that for any such separator S there exists a hypergraph edge cut C on H_{TRSV} that is at most twice the size. The inclusion of any vertex $Z_{ij} \in S$, disconnects this vertex from the rest of the graph and does not disconnect any other path between two vertices in $\mathbf{Z}$, since such dependency paths all go through the reduction tree and $\mathbf{x}$. For each such vertex ($Z_{ij} \in S$) we add edges e_i and e_j from E_{TRSV} into the cut C, which completely disconnects Z_{ij} from other vertices in $\mathbf{Z}$ in H_{TRSV}. The inclusion of a separator vertex $\mathbf{x}_i \in S$, disconnects all vertices which are dependencies of x_i (Z_{ij} for $j \in \{1, \ldots, i-1\}$) from all vertices which are dependent on x_i (Z_{ki} for $k \in \{i+1, \ldots, n\}$). In this case, we add edge e_i to C, which has the same effect on H_{TRSV} of disconnecting all vertices in $\mathbf{Z}$ dependent on x_i from all of vertices which are dependencies of x_i in $\mathbf{Z}$. Lastly, for any vertices in S which are part of a reduction tree that contributes to x_i, we add the hyperedge e_i to cut C, disconnecting all dependencies of x_i from its dependants. Thus, C is a hyperedge cut of H_{TRSV} since each hyperedge in H corresponds to a unique partition of edges in $G_{\text{TRSV}}(n)$ (including edges in the reduction trees), and we have disconnected all hyperedges corresponding to the set of edges in $G_{\text{TRSV}}(n)$ which were disconnected by separator S. Therefore, since disconnecting the edges adjacent to S in $G_{\text{TRSV}}(n)$ broke all paths between some two partitions of vertices, so must the hyperedge cut C in H_{TRSV}.

Further, the cut C which we have thus constructed is at most twice the size of S, since we added at most two edges to C for each vertex in S. By Theorem 5.2, any $\frac{1}{q}$-balanced cut of a 2-dimensional lattice hypergraph is of size $\Omega(n/\sqrt{q})$. Therefore, any vertex separator must be of size at least $\chi_q(G_{\text{TRSV}}(n)) = \Omega(n/\sqrt{q})$. □

Theorem 6.2. *Any parallelization of any dependency graph $G_{\text{TRSV}}(n)$ where two processors compute $\lfloor n^2/2q \rfloor$ elements of $\mathbf{Z}$ (for some $2 \leq q \ll n$) must incur a communication cost of*

$$W_{\text{TRSV}} = \Omega\left(n/\sqrt{q}\right).$$

Proof. Let G be any dependency graph $G_{\text{TRSV}}(n)$ for Algorithm 1. Every vertex in G that has an outgoing edge to a vertex computed by a different processor (different color) must be communicated. Since two processors compute $\lfloor n^2/q \rfloor$ vertices of $\mathbf{Z}$, the communicated set can be bounded below by the size of a $\frac{1}{q}$-balanced separator of $\mathbf{Z}$ within G_{TRSV}. By application of Lemma 6.1, the size of any such separator is at least $\Omega(n/\sqrt{q})$. □

Theorem 6.3. *Any parallelization of any dependency graph $G_{\text{TRSV}}(n)$ where two of p processors compute $\lfloor n^2/2p \rfloor$ elements of $\mathbf{Z}$ incurs the following computation (F), bandwidth (W), and latency (S) costs, for some $b \in [1, n]$,*

$$F_{\text{TRSV}} = \Omega\left(n \cdot b\right), \qquad W_{\text{TRSV}} = \Omega\left(n\right), \qquad S_{\text{TRSV}} = \Omega\left(n/b\right),$$

and furthermore,

$$F_{\text{TRSV}} \cdot S_{\text{TRSV}} = \Omega\left(n^2\right).$$

Proof. Let G be any dependency graph $G_{\text{TRSV}}(n)$ for Algorithm 1. We note that the computation of x_i for $i \in \{1, \ldots, n\}$ requires the computation of Z_{jk} for $j, k \in \{1, \ldots, i\}$ with $k < j$. Furthermore, no element Z_{lm} for $l, m \in \{i+1, \ldots, n\}$ with $l < m$ may be computed until x_i is computed. Consider any subpath $\mathcal{R} \subset \mathcal{P}$ of the dependency path $\mathcal{P} = \{x_1, \ldots, x_n\}$. We recall that the bubble $\beta(G, \mathcal{R}) = (V_\beta, E_\beta)$ around $\mathcal{R}$ is the set of all computations that depend on an element of $\mathcal{R}$ or influence an element of $\mathcal{R}$. Evidently, if $\mathcal{R} = \{x_i, \ldots, x_j\}$, the bubble includes vertices corresponding to a subtriangle of $\mathbf{Z}$, namely, $Z_{kl} \in V_\beta$ for $k, l \in \{i, \ldots, j\}$ with $l < k$. Therefore, $\beta(G, \mathcal{R})$ is isomorphic to $G_{\text{TRSV}}(|\mathcal{R}|)$, which implies that $|V_\beta| = \Theta(|\mathcal{R}|^2)$ and by Lemma 6.1, we have $\chi_q(\beta(G, \mathcal{R})) = \Omega(|\mathcal{R}|/\sqrt{q})$. Since the bubbles for TRSV are 2-dimensional we apply Corollary 4.2 with $d = 2$ to obtain, for some $b \in [1, n]$,

$$F_{\text{TRSV}} = \Omega\left(n \cdot b\right), \qquad W_{\text{TRSV}} = \Omega\left(n\right), \qquad S_{\text{TRSV}} = \Omega\left(n/b\right).$$

□

6.1.2 Attainability

The lower bounds presented above for triangular solve, are attained by the communication-efficient execution blocking schedule suggested in Papadimitriou and Ullman [17]. Algorithm 2 below uses this blocking schedule with blocking factor b to compute the triangular solve. Our algorithm is similar to the wavefront algorithm given by Heath et al. [11].

The parallel Algorithm 2 can be executed using $p = n/b$ processors. Let processor p_l for $l \in \{1, \ldots, n/b\}$ initially own L_{ij}, y_j for $i \in \{1, \ldots, n\}$, $j \in \{(l-1)b + 1, \ldots, lb\}$. Processor p_l performs parallel loop iteration l at each step of Algorithm 2. Since it owns

Algorithm 2 Parallel triangular solve (TRSV) algorithm

$\mathbf{x} = \text{TRSV}(\mathbf{L}, \mathbf{y}, n)$

```
1  x = y
2  for k = 1 to n/b
3      // Each processor p_l executes a unique iteration of below loop
4      parallel for l = max(1, 2k − n/b) to k
5          if l > 1
6              Receive length b vector x[(2k − l − 1)b + 1 : (2k − l)b] from processor p_{l−1}
7          for i = (2k − l − 1)b + 1 to (2k − l)b
8              for j = (l − 1)b + 1 to min(i − 1, lb)
9                  x_i = (x_i − L_ij · x_j)
10             if k = l
11                 x_i = x_i / L_ii
12         if l < n/b
13             Send length b vector x[(2k − l − 1)b + 1 : (2k − l)b] to processor p_{l+1}
14     parallel for l = max(1, 2k + 1 − n/b) to k
15         if l > 1
16             Receive length b vector x[(2k − l)b + 1 : (2k − l + 1)b] from processor p_{l−1}
17         for i = (2k − l)b + 1 to (2k − l + 1)b
18             for j = (l − 1)b + 1 to lb
19                 x_i = (x_i − L_ij · x_j)
20         if l < n/b
21             Send length b vector x[(2k − l)b + 1 : (2k − l + 1)b] to processor p_{l+1}
```

the necessary panel of $\mathbf{L}$ and vector part x_j, no communication is required outside the vector send/receive calls listed in the code. So at each iteration of the outer loop at least one processor performs $O(b^2)$ work, and $2b$ data is sent, requiring 2 messages. Therefore, this algorithm achieves the following costs,

$$F_{\text{TRSV}} = O(nb), \quad W_{\text{TRSV}} = O(n), \quad S_{\text{TRSV}} = O(n/b),$$

which attains our communication lower bounds in Theorems 6.2 and 6.3 for any $b \in \{1, n\}$. Parallel TRSV algorithms in current numerical libraries such as Elemental [18] and ScaLAPACK [6] employ algorithms that attain our lower bound, modulo an extra $O(\log(p))$ factor on the latency cost, due to their use of collectives for communication rather than the point-to-point communication in our wavefront TRSV algorithm.

6.2 Gaussian elimination

In this section, we show that the Gaussian elimination algorithm has 3-dimensional bubble-growth and dependency graphs which satisfy the path expansion properties necessary for the application of Corollary 4.2 with $d = 3$. We consider factorization of a symmetric matrix via Cholesky, as well as Gaussian elimination of a dense nonsymmetric matrix. We show that these factorizations of n-by-n matrices form an intermediate 3D tensor $\mathbf{Z}$ such that $Z_{ijk} \in \mathbf{Z}$ for $i > j > k \in \{1, \ldots, n\}$ and Z_{ijk} is dependent on each Z_{lmn} for $l > m > n \in \{1, \ldots j - 1\}$. We assume that a fast matrix multiplication algorithm is not used, though we conjecture that our analysis can be extended to account for potential use of Strassen's matrix multiplication algorithm and likely other fast algorithms for multiplication.

6.2.1 Cholesky factorization

The Cholesky factorization of a symmetric positive definite matrix $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

for a lower-triangular matrix $\mathbf{L}$. A simple sequential algorithm for Cholesky factorization is given in Algorithm 3. We introduced an intermediate tensor $\mathbf{Z}$, whose elements must be computed during any execution of the Cholesky algorithm (although $\mathbf{Z}$ itself need not be stored explicitly in an actual implementation). We note that the Floyd-Warshall [10, 25] all-pairs shortest-path graph algorithm has the same dependency structure as Cholesky for undirected graphs (and Gaussian Elimination for directed graphs), so our lower bounds may be easily extended to this case. However, interestingly our lower bounds for this graph algorithm are not valid for the all-pairs shortest-paths problem in general, which may alternatively be solved via path-doubling (a technique which naively incurs an extra computational cost, but may be augmented to have the same asymptotic costs as matrix multiplication, as shown by Tiskin [23]).

Algorithm 3 Cholesky factorization algorithm

$\mathbf{L} = \text{CHOLESKY}(\mathbf{A}, n)$

1 **for** $j = 1$ **to** n
2 $\quad L_{jj} = \sqrt{A_{ij} - \sum_{k=1}^{j-1} L_{jk} \cdot L_{jk}}$
3 $\quad$ **for** $i = j + 1$ **to** n
4 $\quad\quad$ **for** $k = 1$ **to** $j - 1$
5 $\quad\quad\quad Z_{ijk} = L_{ik} \cdot L_{jk}$
6 $\quad\quad L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} Z_{ijk})/L_{jj}$

6.2.2 LU factorization

The LU factorization of a square matrix $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U},$$

for a lower-triangular matrix $\mathbf{L}$ and a unit-diagonal upper triangular matrix $\mathbf{U}$ (we make $\mathbf{U}$ rather than $\mathbf{L}$ have a unit-diagonal for notational convenience). A simple non-pivoted algorithm for LU factorization is given in Algorithm 4. Within the computation of

Algorithm 4 LU factorization algorithm

$\mathbf{L}, \mathbf{U} = \text{LU}(\mathbf{A}, n)$

1 **for** $j = 1$ **to** n
2 $\quad$ **for** $i = 1$ **to** $j - 1$
3 $\quad\quad$ **for** $k = 1$ **to** $i - 1$
4 $\quad\quad\quad \bar{Z}_{jik} = L_{ik} \cdot U_{kj}$
5 $\quad\quad U_{ij} = A_{ij} - \sum_{k=1}^{i-1} \bar{Z}_{jik}$
6 $\quad L_{jj} = A_{ij} - \sum_{k=1}^{j-1} L_{jk} \cdot U_{kj}$
7 $\quad$ **for** $i = j + 1$ **to** n
8 $\quad\quad$ **for** $k = 1$ **to** j
9 $\quad\quad\quad Z_{ijk} = L_{ik} \cdot U_{kj}$
10 $\quad\quad L_{ij} = (A_{ij} - \sum_{k=1}^{j-1} Z_{ijk})/L_{jj}$

LU factorization, the $n^3/6$ intermediate vertices in $\mathbf{Z}$ within Algorithm 4 are analogous to the intermediate vertices of the Cholesky computation of the previous section. The vertices designated as $\bar{\mathbf{Z}}$ in the LU computation are ignored in our further analysis. Ignoring vertices does not invalidate our lower bounds because they can only necessitate more work and communication.

6.2.3 Lower bounds

We note that the summations on lines 2 and 6 of Algorithm 3 as well as lines 6 and 10 of Algorithm 4 can be computed via any summation order (and will be computed in different orders in different parallel algorithms). This implies that the summed vertices are connected in any dependency graph $G_{\text{GE}}(n)$, but the connectivity structure may be different. We define a 3-dimensional lattice hypergraph $H_{\text{GE}} = (V_{\text{GE}}, E_{\text{GE}})$ for the algorithm which allows us to obtain a lower bound for any possible summation order, as

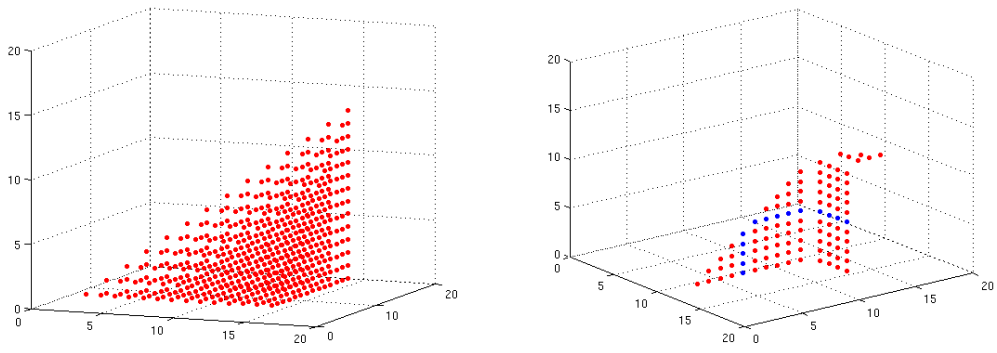
$$\begin{aligned} V_{\text{GE}} =& \{Z_{ijk} : i, j, k \in \{1, \ldots, n\}, i > j > k\}, \\ E_{\text{GE}} =& \{e_{i,j} : i, j \in \{1, \ldots, n\} \text{ with } i > j\} \text{ where} \\ & e_{i,j} = \{Z_{ijk} : k \in \{1, \ldots, j-1\}\} \cup \{Z_{ikj} : k \in \{j+1, \ldots, i-1\}\} \cup \{Z_{kij} : k \in \{i+1, \ldots, n\}\} \end{aligned}$$

Figure 3(a) displays the intermediate vertices of $H_{\text{GE}}(16)$. We enumerate the set of hyperedges E_{GE} via elements $e_{i,j} \in E_{\text{GE}}$, $i, j \in \{1, \ldots, n\}$ with $i > j$.

Lemma 6.4. *Any vertex separator S within dependency graph $G_{\text{GE}}(n)$ that subdivides the intermediate vertices $\mathbf{Z}$ into two sets of size at least $\lfloor n^3/3q \rfloor$ (where $2 \leq q \ll n$) must have size at least*

$$\chi_q(G_{\text{GE}}(n)) = \Omega\left(n^2/q^{2/3}\right).$$

Proof. We show that for any such separator S in G_{GE}, it is possible to construct a hyperedge cut C of H_{GE} which consists of at most $3|S|$ hyperedges. The separator S may include vertices in $\mathbf{Z}$, in $\mathbf{L}$, or in a reduction tree that contributes to an entry in $\mathbf{L}$. In the first case, if S includes an entry Z_{ijk}, then this entry is disconnected entirely from the graph, while the connectivity of other vertices in $\mathbf{Z}$ is



(a) (b)

Figure 3: These diagrams show (a) the vertices Z_{ijk} in V_{GE} with $n = 16$ and (b) the hyperplane x_{12} and hyperedge $e_{12,6}$ on H_{GE}.

not affected. For each such entry $Z_{ijk} \in S$ we add edges $e_{i,j}$, $e_{i,k}$ and $e_{j,k}$ to C, effectively disconnecting Z_{ijk} within the hypergraph H_{GE}. In the latter two cases, if S includes entry L_{ij} or an entry to a reduction tree that contributes to L_{ij}, we add edge $e_{i,j}$ to C. Including the entry L_{ij} or a intermediate in a reduction tree which contributes to it disconnects vertices which are dependent on L_{ij} from the dependencies thereof. These dependencies are encoded in the hypergraph H_{GE} by the edge $e_{i,j}$, the removal of which serves to break all possible paths that could have gone through L_{ij} or the reduction tree in H_{GE}. Now we ascertain that C is a hyperedge cut of H_{GE} since each hyperedge in H corresponds to a unique partition of edges in $G_{\mathrm{GE}}(n)$ (including edges in the reduction trees), and we have disconnected all hyperedges corresponding to the set of edges in $G_{\mathrm{GE}}(n)$ which were disconnected by separator S. Therefore, since disconnecting the edges adjacent to S broke all paths between some two partitions of vertices in $G_{\mathrm{GE}}(n)$, the hyperedge cut C must disconnect the same partitions in H_{GE}.

For each vertex in S, we have added at most 3 edges to the hyperedge cut C, and have disconnected the same or larger sets of vertices within the hypergraph in each case. By Theorem 5.2, a $\frac{1}{q}$-balanced cut of the vertices $\mathbf{Z}$ in H_{GE} is of size $\Omega(n^2/q^{2/3})$. Therefore, any vertex separator on a $G_{\mathrm{GE}}(n)$ must be of size at least $\chi_q(G_{\mathrm{GE}}(n)) = \Omega(n^2/q^{2/3})$. □

Theorem 6.5. *Any parallelization of any dependency graph $G_{\mathrm{GE}}(n)$, where two processors each compute $\lfloor n^3/3q \rfloor$ elements of $\mathbf{Z}$ (V_{GE}), must incur a communication of*

$$W_{\mathrm{GE}} = \Omega\left(n^2/q^{2/3}\right).$$

Proof. For any $G_{\mathrm{GE}}(n)$, every vertex that has an outgoing edge to a vertex computed by a different processor (different color) must be communicated. Since two processors each compute $\lfloor n^3/3q \rfloor$ elements of $\mathbf{Z}$, the communicated set can be bounded below by the size of a $\frac{1}{q}$-balanced separator of the vertices $\mathbf{Z}$ in $G_{\mathrm{GE}}(n)$. By Lemma 6.4, the size of any such separator is $\Omega(n^2/q^{2/3})$. □

Theorem 6.6. *Any parallelization of any dependency graph $G_{\mathrm{GE}}(n)$ in which two of p processors compute $\lfloor n^3/3p \rfloor$ vertices of $\mathbf{Z}$ incurs the following computation (F), bandwidth (W), and latency (S) costs, for some $b \in [1, n]$,*

$$F_{\mathrm{GE}} = \Omega\left(n \cdot b^2\right), \qquad W_{\mathrm{GE}} = \Omega\left(n \cdot b\right), \qquad S_{\mathrm{GE}} = \Omega\left(n/b\right),$$

and furthermore,

$$F_{\mathrm{GE}} \cdot S_{\mathrm{GE}}^2 = \Omega\left(n^3\right), \qquad W_{\mathrm{GE}} \cdot S_{\mathrm{GE}} = \Omega\left(n^2\right).$$

Proof. Let G be a dependency graph of $G_{\mathrm{GE}}(n)$. We note that the computation of L_{ii} for $i \in \{1, \ldots, n\}$ requires the computation of Z_{lmk} for $l, m, k \in \{1, \ldots, i\}$ with $l > m > k$. Furthermore, no element Z_{srt} for $s, r, t \in \{i+1, \ldots, n\}$ with $s > r > t$ can be computed until L_{ii} is computed. Consider any subpath $\mathcal{R} \subset \mathcal{P}$ of the dependency path $\mathcal{P} = \{L_{11}, \ldots, L_{nn}\}$. Evidently, if $\mathcal{R} = \{L_{ii}, \ldots, L_{j+1,j+1}\}$, the bubble $\beta(G, \mathcal{R}) = (V_\beta, E_\beta)$ includes vertices corresponding to a subcube of $\mathbf{Z}$, namely $Z_{klm} \in V_\beta$ for $k, l, m \in \{i, \ldots, j\}$ with $k > l > m$. Therefore, $\beta(G, \mathcal{R})$ is isomorphic to $G_{\mathrm{GE}}(|\mathcal{R}|)$, which implies that $|V_\beta| = \Theta(|\mathcal{R}|^3)$ and by Lemma 6.4, we have $\chi_q(\beta(G, \mathcal{R})) = \Theta(|\mathcal{R}|^2/q^{2/3})$. Since we have 3-dimensional bubbles with 2-dimensional cross-sections, we apply Corollary 4.2 with $d = 3$ to obtain, for some $b \in [1, n]$,

$$F_{\mathrm{GE}} = \Omega\left(n \cdot b^2\right), \qquad W_{\mathrm{GE}} = \Omega\left(n \cdot b\right), \qquad S_{\mathrm{GE}} = \Omega\left(n/b\right).$$

□

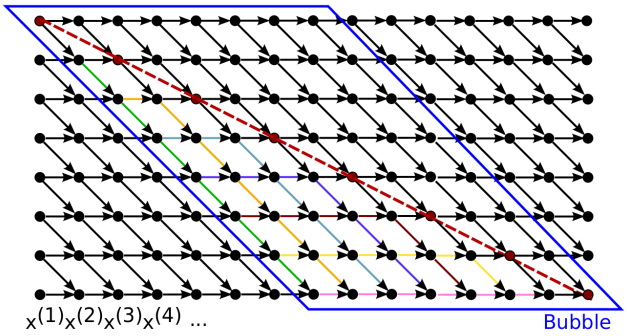


Figure 4: Depiction of the bubble (blue parallelogram) along a dependency path (red dashed path) within a Krylov basis computation on a 1-dimensional mesh (2-point stencil). Edges within the bubble are colored according to the hypergraph edge to which they correspond.

6.2.4 Attainability

The lower bounds presented in the previous section are attained on p processors for $b \approx n/\sqrt{p}$ by '2D algorithms', which utilize a blocked matrix layout and are employed by most standard parallel libraries (including Elemental [18] and ScaLAPACK [6]). The BSP [16] algorithms presented by Tiskin [16] for LU (without pivoting and with pairwise pivoting [21]) and QR factorization with Givens rotations, match the lower bounds in Theorem 6.5 and Theorem 6.6 for any $b \in [n/p^{2/3}, n/\sqrt{p}]$. Therefore, Tiskin's algorithms can lower the bandwidth cost with respect to 2D algorithms by a factor of up to $p^{1/6}$, at the cost of raising latency by the same factor. Similarly, 2.5D algorithms [20] for LU factorization without pivoting and with tournament pivoting [8] also lower bandwidth cost by a factor of up to $p^{1/6}$ by sacrificing latency cost. 2.5D algorithms are practical and can improve upon the performance of standard 2D algorithms so long as the matrix is large enough to amortize synchronization overheads. Therefore, the latency-bandwidth trade-off is of particular importance for this problem.

We note that the 3D parallel LU algorithm given by Irony and Toledo [12], a major motivation for some of the communication-efficient algorithms in the last paragraph, is not optimal in our model, because it does not minimize bandwidth cost along the critical path, but only communication volume. This suboptimality is justified by the more positive performance results for 2.5D algorithms collected in [20] and [19], with respect to the performance observed by the LU implementation of Irony and Toledo [12].

6.3 Krylov basis computation

We consider the s-step Krylov subspace basis computation

$$\mathbf{x}^{(l)} = \mathbf{A} \cdot \mathbf{x}^{(l-1)},$$

for $l \in \{1, \ldots, s\}$ and $\mathbf{x}^{(0)}$ given as input, where the graph of the symmetric sparse matrix $\mathbf{A}$ is a $(2m+1)^d$-point stencil (with $m \geq 1$), i.e., d-dimensional n-by-$\cdots$-by-n mesh T, and each entry in $\mathbf{A}$ represents an interaction between vertices $v_{i_1,\ldots,i_d}, w_{j_1,\ldots,j_d} \in T$, such that for $k \in \{1, \ldots, d\}$, $i_k, j_k \in \{1, \ldots, n\}, |j_k - i_k| \leq m$. Thus, matrix $\mathbf{A}$ and vectors $\mathbf{x}^{(l)}$, $l \in \{0, \ldots, s\}$, have dimension n^d, and $\mathbf{A}$ has $\Theta(m^d)$ nonzeros per row/column. We note that the dependency structure of this computation is analogous to direct force evaluation in particle simulations and the Ford-Fulkerson shortest-path algorithm, which may be expressed as sparse-matrix times vector multiplication but on different algebraic semirings.

Theorem 6.7. *Any parallel execution of an s-step Krylov subspace basis computation for a $(2m+1)^d$-point stencil, for $m \geq 1$, on a d-dimensional mesh with $d \ll s$, requires the following computational, bandwidth, and latency costs for some $b \in \{1, \ldots s\}$,*

$$F_{\mathrm{Kr}} = \Omega\left(m^d \cdot b^d \cdot s\right), W_{\mathrm{Kr}} = \Omega\left(m^d \cdot b^{d-1} \cdot s\right), S_{\mathrm{Kr}} = \Omega\left(s/b\right).$$

and furthermore,

$$F_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^d = \Omega\left(m^d \cdot s^{d+1}\right), \quad W_{\mathrm{Kr}} \cdot S_{\mathrm{Kr}}^{d-1} = \Omega\left(m^d \cdot s^d\right).$$

Proof. In the following analysis, we will discard factors of d, as we assume $d \ll s$ (d is a constant), which is reasonable for most problems of interest ($d \in \{2, 3\}$), but some assumptions in this analysis may need to be revisited if more precise consideration of high-dimensional meshes is desired. We let $G_{\mathrm{Kr}} = (V_{\mathrm{Kr}}, E_{\mathrm{Kr}})$ be the dependency graph of the s-step Krylov subspace basis computation defined above. We index the vertices $V_{\mathrm{Kr}} \ni v_{i_1,\ldots,i_d,l}$ with $d+1$ coordinates, each corresponding to the computation of an intermediate vector element $x_k^{(l)}$ for $l \in \{1, \ldots, s\}$ and $k = \sum_{j=1}^{d} i_j n^{j-1}$ (this assumes the lexicographical ordering of $\{1, \ldots, n\}^d$). For each edge $(v_{i_1,\ldots,i_d}, w_{j_1,\ldots,j_d})$ in T and each $l \in \{1, \ldots, s\}$, there is an edge $(v_{i_1,\ldots,i_d,l-1}, w_{j_1,\ldots,j_d,l})$ in E_{Kr}.

Consider the following path $\mathcal{P}$ and any subpath $\mathcal{R}$, where $|\mathcal{R}| = r \geq 3$,

$$\mathcal{P} = \{v_{1,\ldots,1,1}, \ldots, v_{1,\ldots,1,s}\}$$
$$\mathcal{P} \supset \mathcal{R} = \{v_{1,\ldots,1,h+1}, \ldots, v_{1,\ldots,1,h+r}\}.$$

The dependency bubble $\beta(G_{\rm Kr},\mathcal{R})=(V_\beta,E_\beta)$ includes

$$V_\beta=\big\{v_{i_1,\ldots,i_d,i_{d+1}}:i_j\leq m\cdot\min(i_{d+1}-h-1,h+r-i_{d+1}),j\in\{1,\ldots,d\},\\ h+1\leq i_{d+1}\leq h+r\big\}.$$

For each $(u_1,\ldots,u_{d+1})\in\{1,\ldots,r-2\}^{d+1}$ we define the block

$$B_{u_1,\ldots,u_{d+1}}=\big\{v_{i_1,\ldots,i_{d+1}}\in V_\beta:i_j\in\{\lceil m/2\rceil(u_j-1)+1,\ldots,\lceil m/2\rceil u_j)\},j\in\{1,\ldots,d\},\\ i_{d=1}=u_{d+1}+h+1\big\}.$$

Thus, each block should contain $\lceil m/2\rceil^d$ vertices on the same level, u_{d+1}. We note that because the breadth of the blocks is $\lceil m/2\rceil$ and the interaction distance (stencil radius) is m, every vertex in $B_{u_1,\ldots,u_j,\ldots,u_{d+1}}$ depends on every vertex in $B_{u_1,\ldots,u_j-1,\ldots,u_{d+1}-1}$ for each $j\in\{1,\ldots,d\}$, as well as on vertices $B_{u_1,\ldots,u_j,\ldots,u_{d+1}-1}$.

We now construct a graph $G'_{\rm Kr}=(V'_{\rm Kr},E'_{\rm Kr})$ corresponding to the connectivity of the blocks within the given bubble $\beta(G_{\rm Kr},\mathcal{R})$, enumerating them on a lattice of breadth $g=\lfloor(r-2)/(d+1)\rfloor$. For each $(u_1,\ldots,u_{d+1})\in\{1,\ldots,g\}^{d+1}$, we have $w_{u_1,\ldots,u_{d+1}}\in V'_{\rm Kr}$ corresponding to block $B_{u_1,\ldots,u_d,t}\subset V_{\rm Kr}$ with $t=\sum_{j=1}^{d+1}u_j$. Each vertex $w_{i_1,\ldots,i_{d+1}}$ is connected to vertices $w_{i_1,\ldots,i_j+1,\ldots,i_{d+1}}$, for $j\in\{1,\ldots,d+1\}$, by edges in $E'_{\rm Kr}$. A representative bubble is shown for $d=1$ in Figure 4, where it is evident that the bubble vertices can be enumerated on a skewed lattice as above.

We transform $G'_{\rm Kr}$ into a hypergraph $H=(V_H,E_H)$, so that for $q\ll s$ a $\frac{1}{q}$-balanced separator of $G'_{\rm Kr}$ is proportional to a $\frac{1}{q}$-balanced hyperedge cut of H. We define $V_H=\{w_{i_1,\ldots,i_{d+1}}\in V'_{\rm Kr}:i_1<i_2<\cdots<i_{d+1}\}$, and the hyperedges E_H correspond to unions of vertices adjacent to disjoint subsets of edges in $G'_{\rm Kr}$. In particular, we define hyperedges $e_{i_1,\ldots,i_d}\in E_H$ for $i_1,\ldots,i_d\in\{1,\ldots,g\}$ with $i_1<\cdots<i_d$, to contain all $w_{j_1,\ldots,j_{d+1}}$ which satisfy $j_1<\cdots<j_{d+1}$ and $\{i_1,\ldots,i_d\}\subset\{j_1,\ldots,j_{d+1}\}$. We can form these hyperedges as unions of edges in $G'_{\rm Kr}$,

$$e_{i_1,\ldots,i_d}\subset\bigcup_{k=1}^{i_1-1}(w_{k,i_1,\ldots,i_d},w_{k+1,i_1,\ldots,i_d})\cup\bigcup_{k=i_1}^{i_2-1}(w_{i_1,k,i_2\ldots,i_d},w_{i_1,k+1,i_2,\ldots,i_d})\cup\ldots\cup\bigcup_{k=i_d}^{g-1}(w_{i_1,\ldots,i_d,k},w_{i_1,\ldots,i_d,k+1})$$

where each pair of vertices in the union corresponds to a unique edge in $G'_{\rm Kr}$. Because these hypergraph edges correspond to disjoint subsets of $E'_{\rm Kr}$, any vertex separator of $G'_{\rm Kr}$ can be transformed into a hyperedge cut C of H of the same size or less formed by taking the hypergraph edges in H to which the vertices in the separator are adjacent to. Further, any such $\frac{1}{q}$-balanced separator of $G'_{\rm Kr}$ cannot be smaller than $\frac{1}{d+1}|C|$ because each vertex is adjacent to no more than $d+1$ edges.

We note that the hypergraph H is a lattice hypergraph of dimension $d+1$ and breadth $g=\lfloor(|\mathcal{R}|-2)/(d+1)\rfloor$. By Theorem 5.2, its $\frac{1}{q}$-balanced hyperedge cut has size $|C|\geq\epsilon_q(H)=\Omega(g^d/q^{d/(d+1)})$. Furthermore, since the $\frac{1}{q}$-balanced separator of $\beta(G'_{\rm Kr},\mathcal{R})$ is at least $\frac{1}{d+1}|C|$

$$\chi_q(\beta(G'_{\rm Kr},\mathcal{R}))=\Omega\left(\frac{g^d}{(d+1)q^{d/(d+1)}}\right)=\Omega(|\mathcal{R}|^d),$$

where the last bound follows since we have $d,q\ll s$.

This lower bound on edge cut size in the block graph $G'_{\rm Kr}$ allows us to obtain a lower bound on the size of any $\frac{1}{q}$-balanced separator of $G_{\rm Kr}$, that is larger by a factor of $\Omega(m^d)$. Consider any $\frac{1}{q}$-balanced separator of $G_{\rm Kr}$ that separates the vertices into three disjoint subsets, the separator Q, and the parts V_1 and V_2. If two vertices $u,v\in V_{\rm Kr}$ are in two different partitions (are of different color), $u\in V_1$ and $v\in V_2$, and are in the same block, all vertices in the d adjacent blocks must have all their vertices entirely in Q (since all vertices in adjacent blocks in $G'_{\rm Kr}$ are adjacent to u and v in $G_{\rm Kr}$). Therefore, the number of blocks which contain vertices of different color is less than $|Q|/m^d$ and therefore small with respect to $|V'_{\rm Kr}|/q$. Therefore, Q should yield $\Omega(|V'_{\rm Kr}|/q)$ blocks which contain vertices that are either in the separator or in V_1 and similarly for V_2. Now, since two blocks $B_1\subset(V_1\cup Q)$ and $B_2\subset(V_2\cup Q)$ which contain non-separator vertices of different color may not be adjacent, there must exist a separator block $B_3\subset Q$ for any path on the lattice between B_1 and B_2. Therefore, Q also induces a separator of $G'_{\rm Kr}$ of size no larger than $|Q|\cdot\lceil m/2\rceil^d$. So, we obtain the following lower bound on the size of a separator of $G_{\rm Kr}$: $\chi_q(\beta(G_{\rm Kr},\mathcal{R}))=\Omega(m^d\cdot\chi_q(\beta(G'_{\rm Kr},\mathcal{R})))=\Omega(m^d|\mathcal{R}|^d)$.

Now, the bubble size is $|V_\beta|=\Omega(m^d|\mathcal{R}|^{d+1})$ and the total length of our main dependency path is $|\mathcal{P}|=s$. By Definition 4.1, $G_{\rm Kr}$ is a (ϵ,σ)-path-expander with $\epsilon(b)=m^db^d$ and $\sigma(b)=m^db^{d+1}$. Therefore, by application of Theorem 4.1, with $k=3$, we obtain the following lower bounds, for some $b\in[3,s]$,

$$F_{\rm Kr}=\Omega\left(m^d\cdot b^d\cdot s\right),W_{\rm Kr}=\Omega\left(m^d\cdot b^{d-1}\cdot s\right),S_{\rm Kr}=\Omega\left(s/b\right).$$

□

6.3.1 Attainability

A parallel communication-avoiding algorithm for computing a Krylov subspace basis, termed 'PA1', is presented in [9]. We note that although PA1 performs redundant computation to lower the parallel latency, the amount of redundant work and reduction in messages

made possible by redundant work are not asymptotically significant, and thus the lower bounds of Theorem 6.7 apply. Computing an s-step Krylov subspace basis with a $(2m+1)^d$-point stencil with block size $b \in \{1, \ldots s\}$ can be accomplished by s/b invocations of PA1 with basis size parameter b. The costs for the overall computation using PA1 are then

$$\begin{aligned} F_{\mathrm{Kr}} &= \frac{s}{b} \cdot O(b^{d+1}m^d) &&= O(m^d \cdot b^d \cdot s), \\ W_{\mathrm{Kr}} &= \frac{s}{b} \cdot O(b^d m^d) &&= O(m^d \cdot b^{d-1} \cdot s), \\ S_{\mathrm{Kr}} &= \frac{s}{b} \cdot O(1) &&= O(s/b), \end{aligned}$$

under the assumption $n/p^{1/d} = O(bm)$. This algorithm therefore attains the lower bounds and lower bound tradeoffs of Theorem 6.7.

7 Conclusion

Our lower bounds showed that many numerical problems which have lattice dependency structure, require execution costs which are independent of the number of processors but dependent on the problem size. Architecturally, our results provided lower bounds on execution time, as a function of synchronization latency (α), communication throughput (β), and clock-speed (γ). The tradeoffs we derive describe the strong scaling limit of Gaussian Elimination and Krylov basis computation in terms of these three quantities. In other words, we obtained bounds on the time it takes for any number of processors to solve a system of linear equations via certain numerical algorithms based on the network and clock speed of each processor. An interesting piece of future work will be to consider Krylov basis computations, which are analogous to the Ford-Fulkerson single-source shortest-paths graph algorithm, on graphs such as expanders and binary trees rather than just stencils (grids), which our graph-based bubble-expansion formulation should allow.

8 Acknowledgements

We would like to thank Satish Rao, Grey Ballard, and Benjamin Lipshitz for particularly useful discussions on topics encompassed within this paper. The first author was supported by a Krell Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. We also acknowledge the support of the US DOE (grants DE-SC0003959, DE-SC0004938, DE-SC0005136, DE-SC0008700, DE-AC02-05CH11231) and DARPA (award HR0011-12-2-0016).

References

[1] A. Aggarwal, A.K. Chandra, and M. Snir. Communication complexity of PRAMs. *Theoretical Computer Science*, 71(1):3 – 28, 1990.

[2] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds (brief announcement). In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, June 2012.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in linear algebra. *SIAM J. Mat. Anal. Appl.*, 32(3), 2011.

[4] E. Bampis, C. Delorme, and J.-C. König. Optimal schedules for d-D grid graphs with communication delays. In *STACS 96*, volume 1046 of *Lecture Notes in Computer Science*, pages 655–666. Springer Berlin Heidelberg, 1996.

[5] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. *Theory of Computing Systems*, 47(4):934–962, 2010.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK user's guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM.

[8] J. Demmel, L. Grigori, and H. Xiang. A Communication Optimal LU Factorization Algorithm. EECS Technical Report EECS-2010-29, UC Berkeley, March 2010.

[9] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical Report UCB/EECS-2007-123, EECS Dept., U.C. Berkeley, Oct 2007.

[10] R.W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345, June 1962.

[11] M.T. Heath and C.H. Romine. Parallel solution of triangular systems on distributed-memory multiprocessors. *SIAM Journal on Scientific and Statistical Computing*, 9(3):558–588, 1988.

[12] D. Irony and S. Toledo. Trading replication for communication in parallel distributed-memory dense solvers. *Parallel Processing Letters*, 71:3–28, 2002.

[13] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017 – 1026, 2004.

[14] H. Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, STOC '81, pages 326–333, New York, NY, USA, 1981. ACM.

[15] L.H. Loomis and H. Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the AMS*, 55:961–962, 1949.

[16] W. F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24:287–297, 1999.

[17] C. Papadimitriou and J. Ullman. A communication-time tradeoff. *SIAM Journal on Computing*, 16(4):639–646, 1987.

[18] J. Poulson, B. Maker, J. R. Hammond, N. A. Romero, and R. van de Geijn. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2):13, 2013.

[19] E. Solomonik, A. Bhatele, and J. Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *Supercomputing, Seattle, WA, USA*, Nov 2011.

[20] E. Solomonik and J. Demmel. Communication-optimal 2.5D matrix multiplication and LU factorization algorithms. In *Springer Lecture Notes in Computer Science, Proceedings of Euro-Par, Bordeaux, France*, Aug 2011.

[21] D.C. Sorensen. Analysis of pairwise pivoting in Gaussian Elimination. *Computers, IEEE Transactions on*, C-34(3):274 –278, March 1985.

[22] A Tiskin. *The Design and Analysis of Bulk-Synchronous Parallel Algorithms*. PhD thesis, University of Oxford, 1998.

[23] A. Tiskin. All-pairs shortest paths computation in the BSP model. *Lecture Notes in Computer Science, Automata, Languages and Programming*, 2076:178–189, 2001.

[24] A. Tiskin. Communication-efficient parallel generic pairwise elimination. *Future Generation Computer Systems*, 23(2):179 – 188, 2007.

[25] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9:11–12, January 1962.